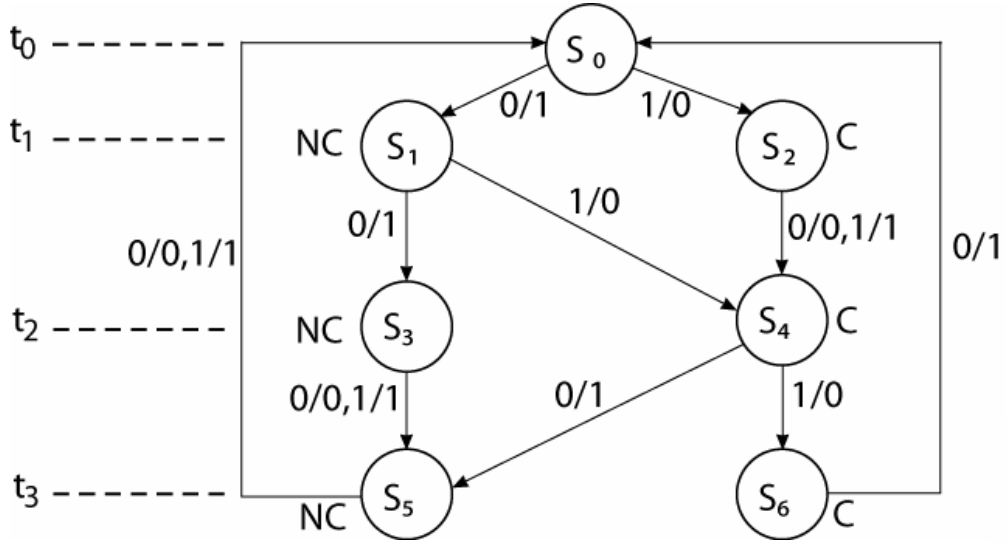
Digital Integrated Circuits

Lecture 4

Jaeyong Chung
System-on-Chips (SoC) Laboratory
Incheon National University

BCD TO EXCESS-3 CODE CONVERTER




```
  0100      0101
+0011     +0011
 0111      1000
```

LSB received first

	NS		Z	
PS	X = 0	X = 1	X = 0	X = 1
S_0	S_1	S_2	1	0
S_1	S_3	S_4	1	0
S_2	S_4	S_4	0	1
S_3	S_5	S_5	0	1
S_4	S_5	S_6	1	0
S_5	S_0	S_0	0	1
S_6	S_0	–	1	–

STATE ASSIGNMENT

- One-Hot State Encoding
  - One Flip-Flop per State
  - Simple Combinational Logic

- Minimal State Encoding
  - Use $\lceil \log_2(\#states) \rceil$ Flip-Flops
  - Assignment of State Codes
    - Impacts Amount of Combinational Logic

STATE ASSIGNMENT GUIDELINES

- Assign Adjacent State Codes
  - States with same NS for given input
  - States that are NS of same state
  - States having same output for given input

	NS		Z	
PS	X = 0	X = 1	X = 0	X = 1
S_0	S_1	S_2	1	0
S_1	S_3	S_4	1	0
S_2	S_4	S_4	0	1
S_3	S_5	S_5	0	1
S_4	S_5	S_6	1	0
S_5	S_0	S_0	0	1
S_6	S_0	–	1	–

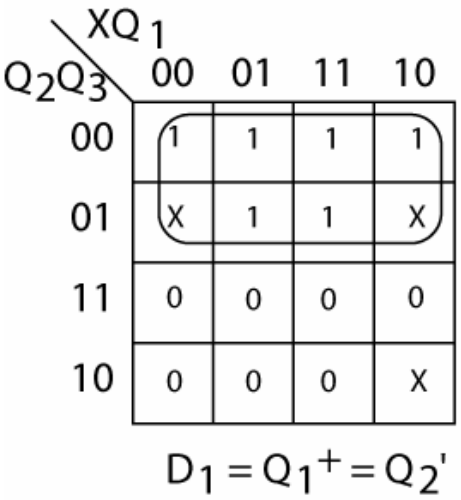
ASSIGNMENT MAP AND TRANSITION TABLE

- State Assignment Guidelines
  - (1,2), (3,4), (5,6), (0,1,4,6), (2,3,5)

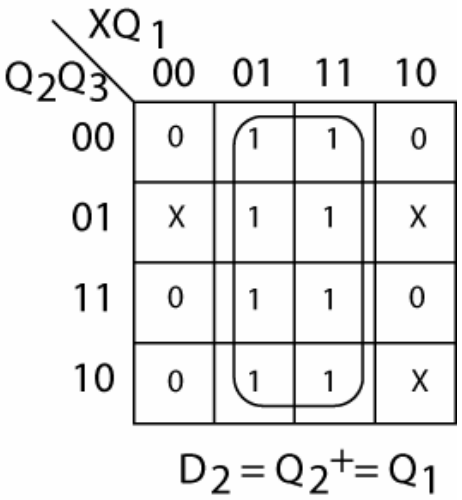
Q_2Q_3 \ Q_1	0	1
00	S_0	S_1
01		S_2
11	S_5	S_3
10	S_6	S_4

	$Q_1^+ Q_2^+ Q_3^+$		Z	
$Q_1Q_2Q_3$	X = 0	X = 1	X = 0	X = 1
000	100	101	1	0
100	111	110	1	0
101	110	110	0	1
111	011	011	0	1
110	011	010	1	0
011	000	000	0	1
010	000	xxx	1	x
001	xxx	xxx	x	x

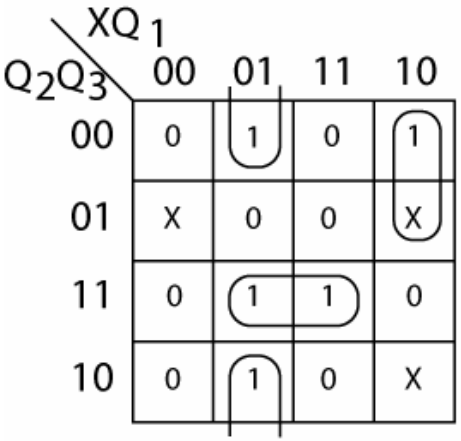
K-MAPS FOR BCD TO EXCESS-3 CONVERTER



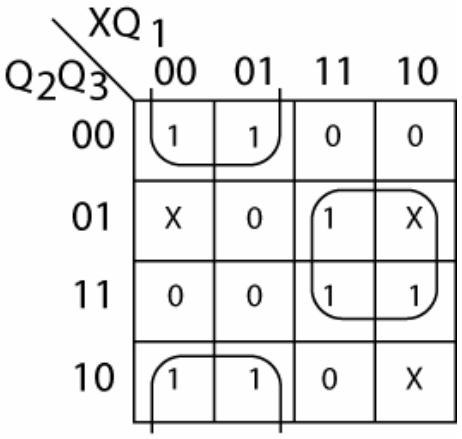
$D_1 = Q_1^+ = Q_2'$



$D_2 = Q_2^+ = Q_1$

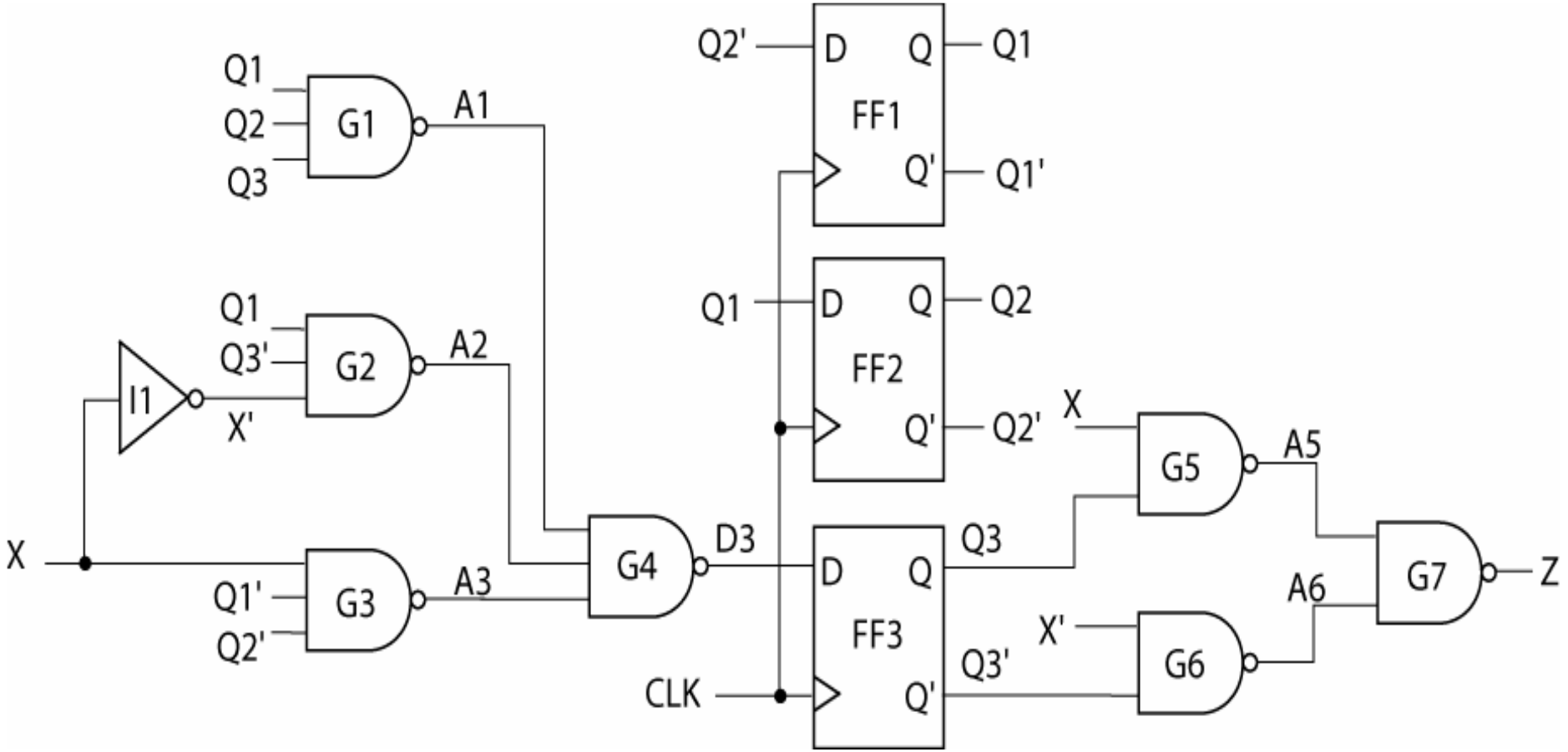


$D_3 = Q_3^+ = Q_1Q_2Q_3 + X'Q_1Q_3' + XQ_1'Q_2'$

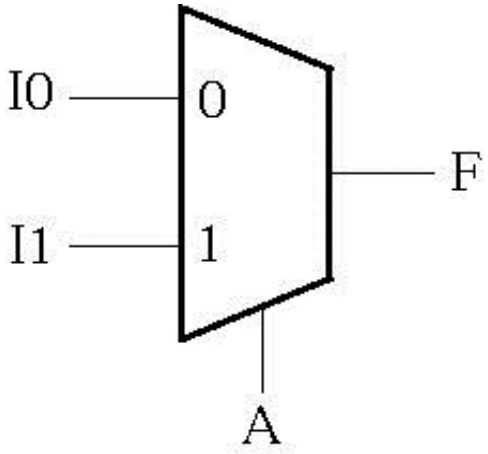


$Z = X'Q_3' + XQ_3$

REALIZATION OF CODE CONVERTER

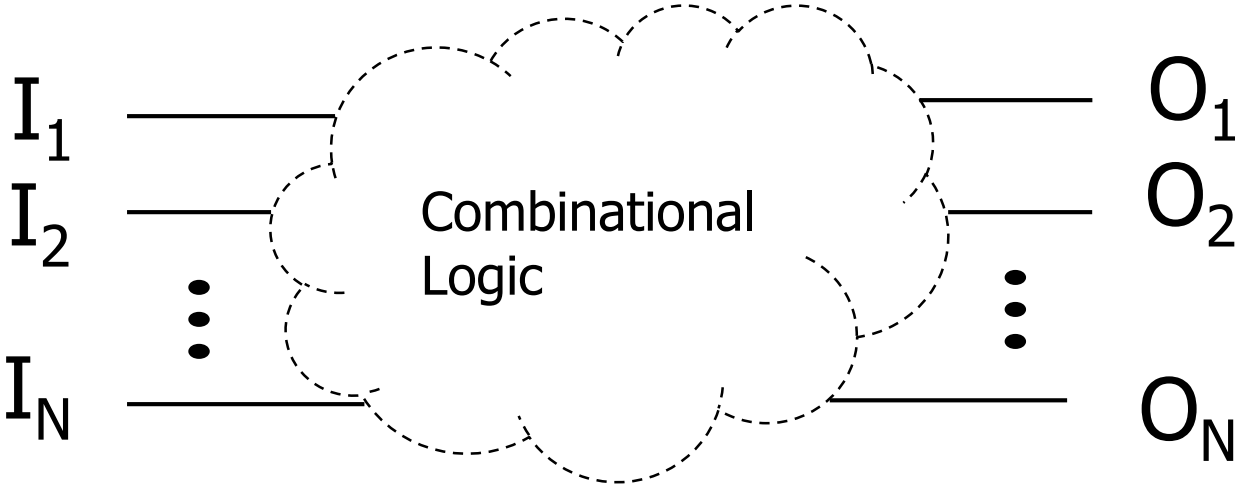


VERILOG MODEL FOR MUX




```
module mux_assign (I0, I1, A, F);
  input I0, I1, A;
  output F;
  wire F;

  assign F = (A) ? I1 : I0;

endmodule
```

ALWAYS BLOCK FOR COMB. LOGIC




```
reg O1;
reg O2;
...
reg ON;
```

1. Declare each output as reg (but no seq. gates will be inferred)

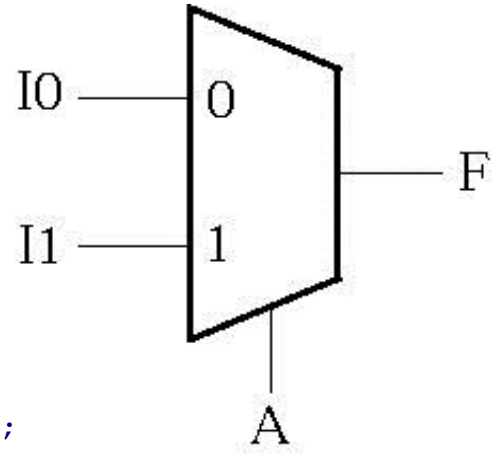
2. Put all inputs into the sensitivity list

```
always @(I1 or I2 or I3 or ... or IN) begin
       O1 = 0; O2 = 0; ... ON = 0;
              ...
              O1 = ... ;
              ...
              O2 =  ... ;
              ...
              ON = ... ;
end
```

3. Initialize each output (Optional).

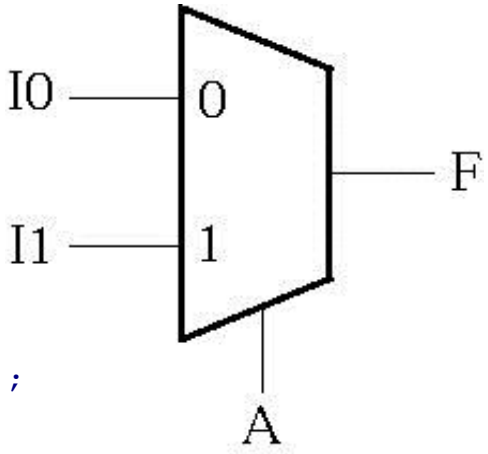
4. Check each output is assigned at least once regardless of inputs

VERILOG MODEL FOR MUX



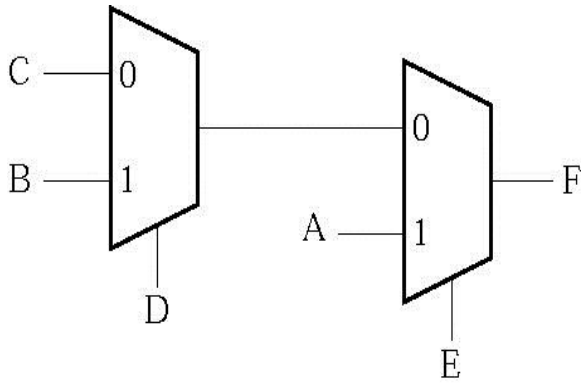

```
module mux_if (I0, I1, A, F);
  input I0, I1, A;
  output F;
  reg F;

  always @ (I0 or I1 or A) begin
          if(A == 1'b0) begin
                    F = I0;
          end else begin
                    F = I1;
          end
  end

endmodule
```

Verilog MODEL FOR MUX

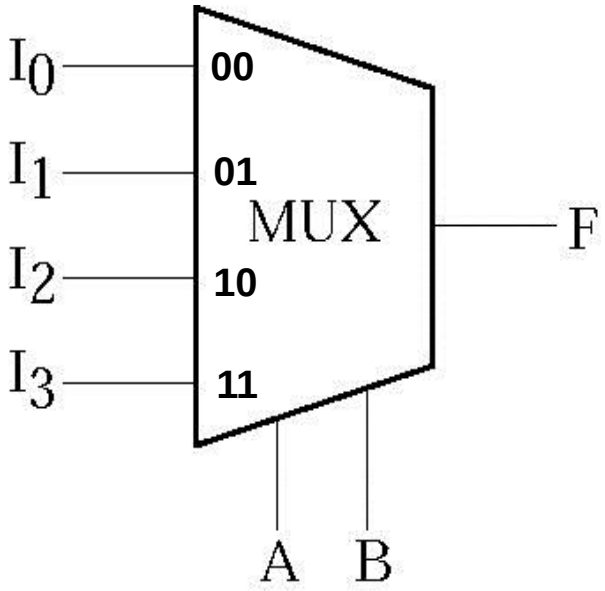



```
module mux_case (I0, I1, A, F);
  input I0, I1, A;
  output F;
  reg F;

  always @ (I0 or I1 or A) begin
     case (A) begin
          1'b0 : F = I0;
          1'b1 : F = I1;
          default :
    endcase
  end

endmodule
```

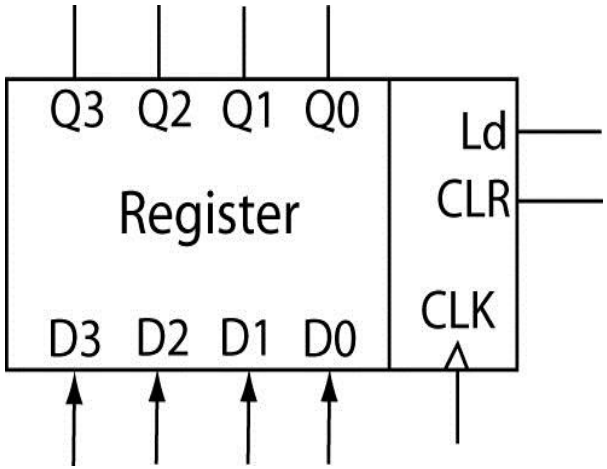
CASCADED 2-TO-1 MUXES



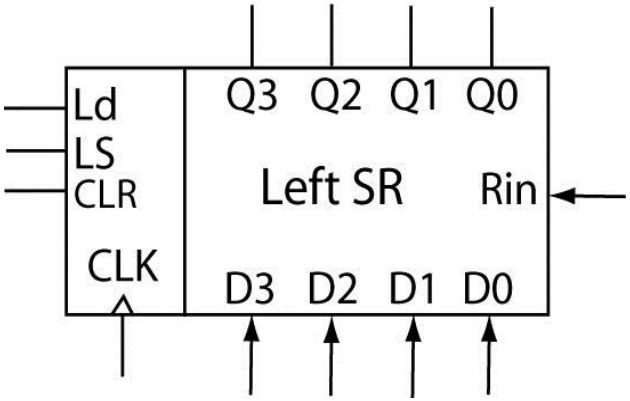
4-TO-1 MUX



REG. WITH SYNCHRONOUS CLEAR AND LOAD

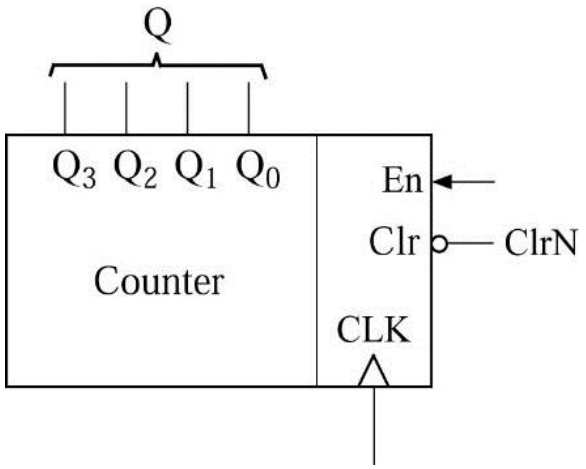


LEFT SHIFT REG. WITH SYNC. CLEAR & LOAD




```
always @(posedge CLK)
begin
  if (CLR == 1'b1) begin
        Q <= 4'b0000;
  end else if (Ld == 1'b1) begin
        Q <= D;
  end else if (Ls == 1'b1) begin
    Q <= {D[2:0], Rin}    // Bit
Concatenation
  end
end
```

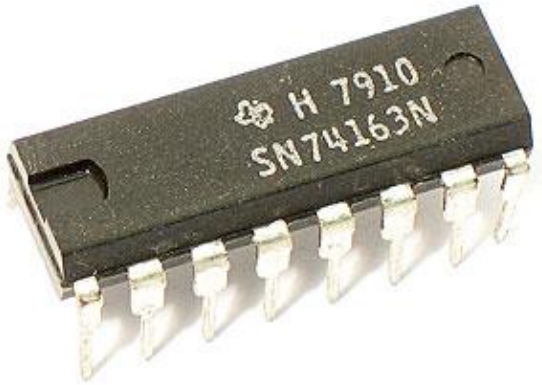
SIMPLE SYNCHRONOUS COUNTER



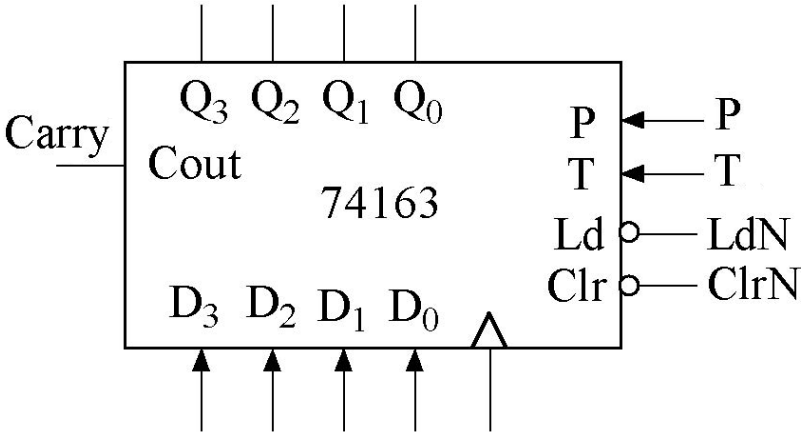

```
always @(posedge CLK)
begin
  if (ClrN == 1'b0) begin
    Q <= 4'b0000;
  end else if (En == 1'b1) begin
    Q <= Q + 1'b1;
  end else begin
    Q <= Q
  end
end
```

74163 COUNTER

- Standard Part
- Available in TTL and CMOS
- 4 Bit Counter



74163 COUNTER

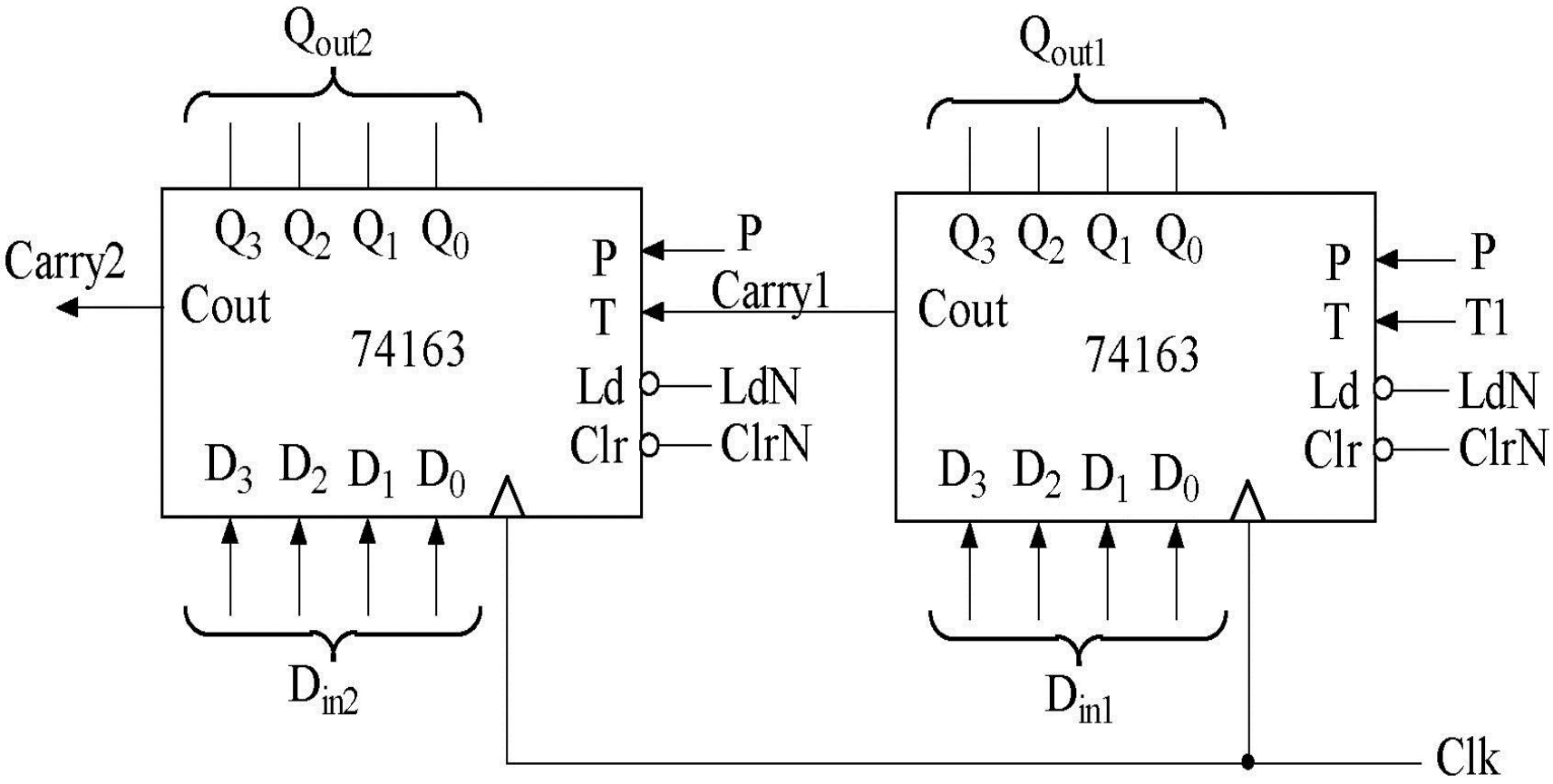


Control Signals			Next State				
ClrN	LdN	P·T	Q_3^+	Q_2^+	Q_1^+	Q_0^+	
0	X	X	0	0	0	0	(clear)
1	0	X	D_3	D_2	D_1	D_0	(parallel load)
1	1	0	Q_3	Q_2	Q_1	Q_0	(no change)
1	1	1	present state + 1				(increment count)

74163 COUNTER

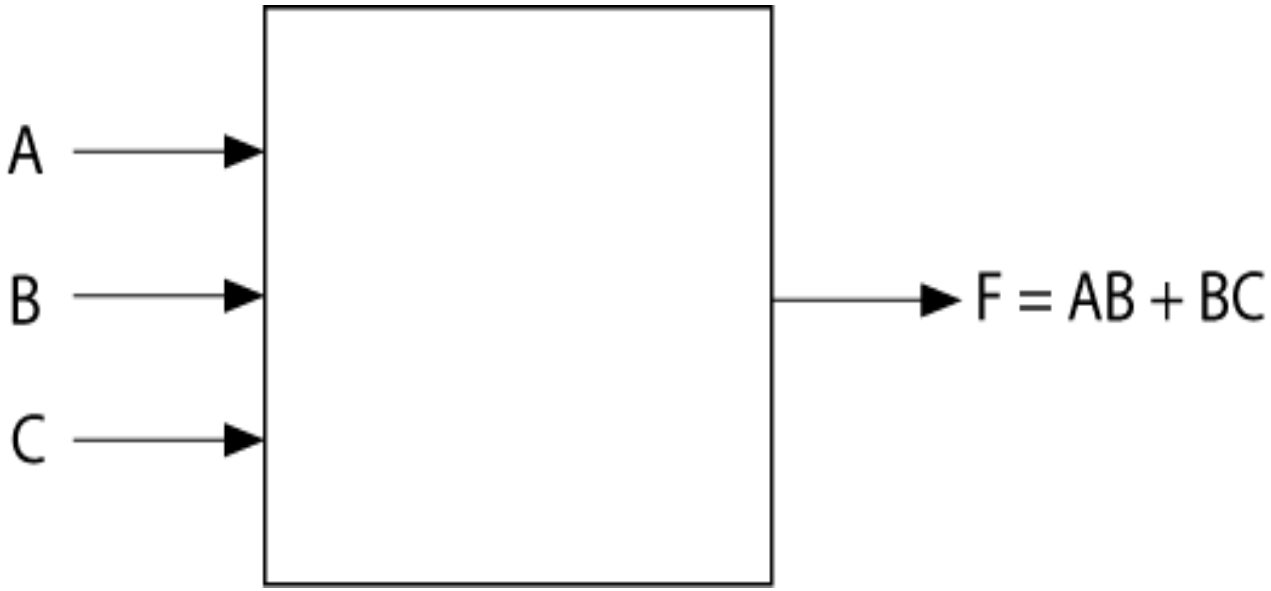
```
module c74163 (D, Clk, ClrN, LdN, P, T, Carry, Q);
  input [3:0] D;
  input       ClrN, LdN, P, T;
  output [3:0] Q;
  output      Carry;
  reg   [3:0] Q;
  reg         Carry;

always @(posedge Clk) begin
  if (ClrN == 1'b0) begin      //ClrN = 0
    Q <= 4'b0;
    Carry <= 1'b0;
  end else begin               //ClrN = 0, LdN = 0
    if (LdN == 1'b0) begin
      Q <= D;
    end else begin
      if( (P && T) == 1'b0) begin       //ClrN = 0, LdN = 1, PT = 0
        Q <= Q;
      end else begin                    //ClrN = 0, LdN = 1, PT = 1
        {Carry, Q} <= Q + 1;
      end
    end
  end
end

endmodule
```

8-BIT COUNTERS FROM 74163



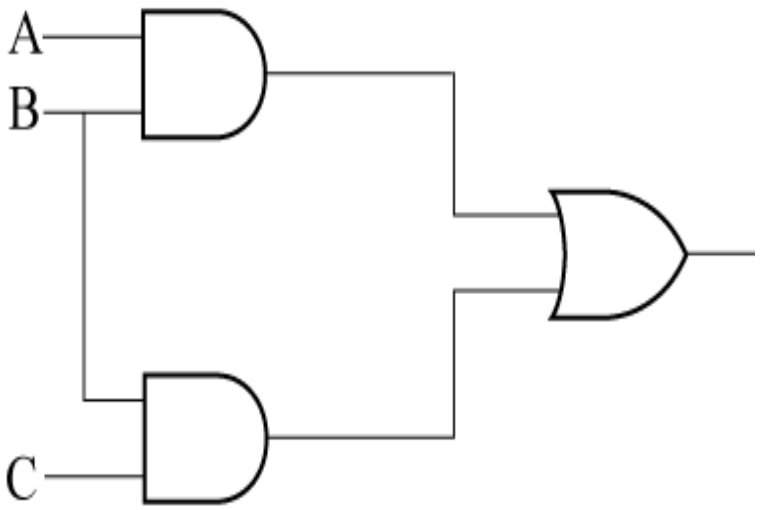
BLOCK DIAGRAM

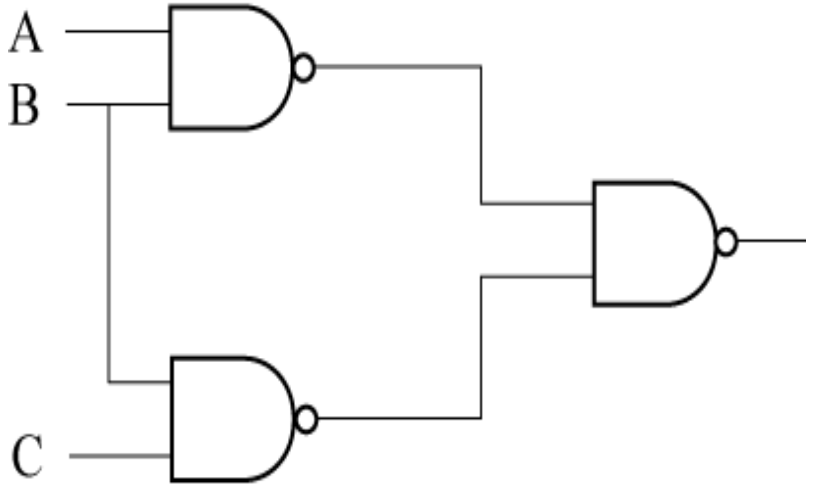
- *A, B, C* as Inputs and *F* = *ab* + *bc* as Output



TWO STRUCTURAL IMPLEMENTATIONS

- *F* = *ab* + *bc* Describes Functionality
  - These 2 Structures Describe How Realized





MULTIPLE LEVELS OF ABSTRACTION

- Verilog Allows Describing Hardware at Different Levels of Abstraction
  - Behavior – No hardware implied
  - Structural – Interconnection of components
- Synthesis Tools convert from high-level to lower level
- State-of-the-art Design Automation Tools
  - Generate Efficient Hardware for Logic and Arithmetic Circuits
  - Memory Structures Often Need Manual Optimizations

CODE

- Using Structural Code
  - Similar to In-Lining Assembly Code in Software
  - Allows Designer to Fully Control Final Design
    - Can Hand Optimize

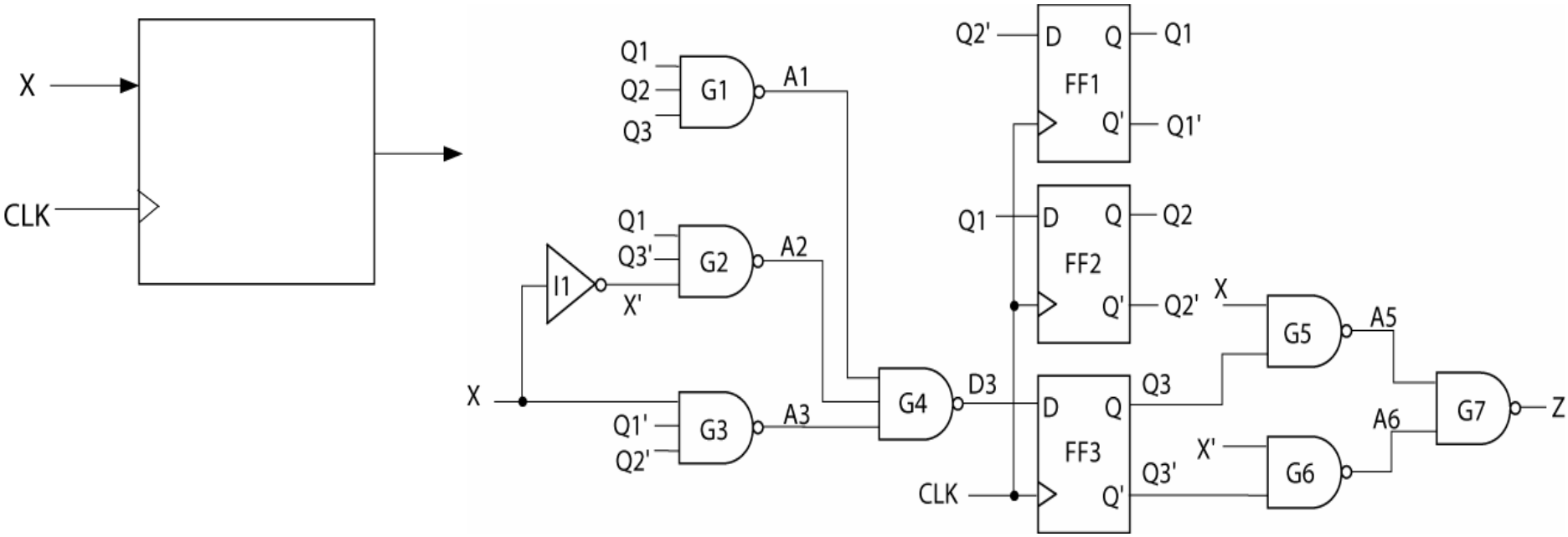
- Using Behavior Code
  - Often Used
  - Less Design Time
    - Important for Time-to-Market
  - Synthesis Tools Quite Good

Simulation/Test Bench

- Test Bench
  - Design Done
    - Need to Test to See whether Meet Design Spec
  - How to Test?
    - Set Initial Conditions
    - Drive Inputs (Possibly All Combinations)
    - Compare Outputs

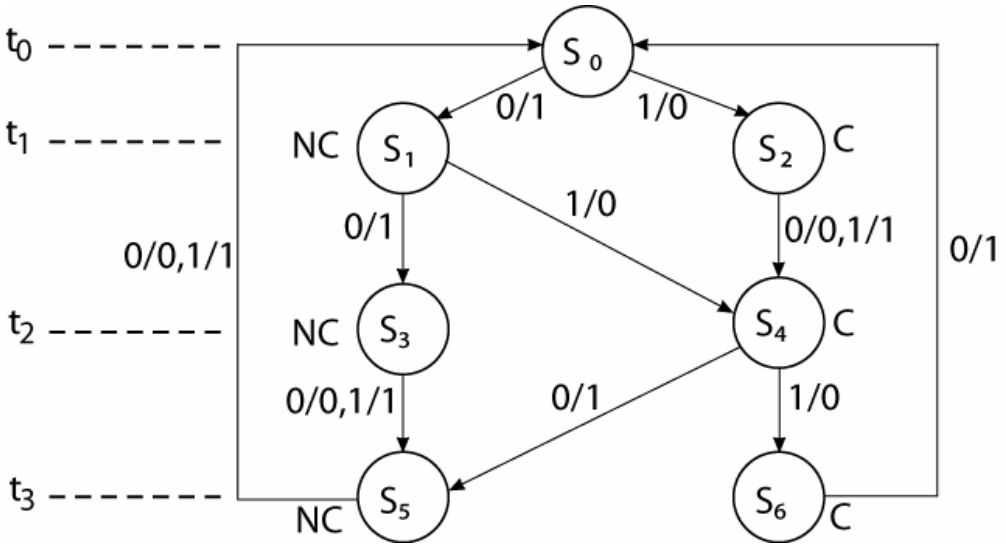
Simulation/Test Bench

- BCD to Excess-3 Code Converter
  - Designed in Chapter 2



Simulation/Test Bench

- BCD to Excess-3 Code Converter
  - Designed in Chapter 2



	NS		Z	
PS	X = 0	X = 1	X = 0	X = 1
S_0	S_1	S_2	1	0
S_1	S_3	S_4	1	0
S_2	S_4	S_4	0	1
S_3	S_5	S_5	0	1
S_4	S_5	S_6	1	0
S_5	S_0	S_0	0	1
S_6	S_0	–	1	–

Simulation/Test Bench

- BCD to Excess-3 Code Converter
  - 4 Different Ways of Implementation
    - Behavioral with Separate Combinational and Sequential Processes
    - Behavioral with Single Process
    - Data Flow with Equations
    - Structural

Simulation/Test Bench

- BCD to Excess-3 Code Converter
  - Sample Code (Parallel)

```
module bcd_to_excess3 (x, clk, z);
  input [3:0] x;
  output [3:0] z;
  reg [3:0] z;

always @ (posedge clk) begin
  z = x + 4'd3;
end
endmodule
```

  - Need to Write Test Bench
    - Set Initial Condition: Reset Registers
    - Drive Input : Drive X

Simulation/Test Bench

- BCD to Excess-3 Code Converter
  - Set Initial Condition
    - initial begin
      ~~
      end
    - initial Block Executed Only Once when Simulation Starts

Simulation/Test Bench

- BCD to Excess-3 Code Converter
  - Set Initial Condition & Drive input

```
module bcd_to_excess3 (x, clk, z);
  input [3:0] x;
  output [3:0] z;
  reg [3:0] z;

always @ (posedge clk) begin
  z = x + 4'd3;
end
endmodule
```

```
initial begin
  x= 0;
  clk=0;
  repeat(9) begin
    #20 x = x + 1'b1;
  end
end
```

Simulation/Test Bench

- BCD to Excess-3 Code Converter
  - Test Bench Module

```
module bcd_to_excess3 (x, clk, z);
  input [3:0] x;
  output [3:0] z;
  reg [3:0] z;

always @ (posedge clk) begin
  z = x + 4'd3;
end
endmodule
```

```
module tb_bcd_to_excess3;  //no input, output ports
reg [3:0] x;
reg clk;
wire [3:0] z;

initial begin
    x = 3'b0;
    clk = 0;
    repeat(9) begin
      #20 x = x + 1'b1;
    end
end

always begin  //clock geneartion
  #5 clk = !clk;
end

bcd_to_excess3 m0_bcd_to_excess3 (.x(x), .clk(clk),
.z(z));

endmodule
```

Simulation/Test Bench

- BCD to Excess-3 Code Converter
  - Test Bench Module

```
module bcd_to_excess3 (x, clk, z);
  input [3:0] x;
  output [3:0] z;
  reg [3:0] z;

always @ (posedge clk) begin
  z = x + 4'd3;
end
endmodule
```

```
module tb_bcd_to_excess3;  //no input, output ports
reg [0:3] x;
reg clk;
wire [0:3] z;

initial begin
    x = 3'b0;
    clk = 0;
    repeat(9) begin
      #20 x = x + 1'b1;
    end
end

always begin  //clock geneartion
  #5 clk = !clk;
end

bcd_to_excess3 m0_bcd_to_excess3 (.x(x), .clk(clk),
.z(z));

initial begin
  $monitor("x = %b, z=%b %b %b %b", x, z[0], z[1], z[2],
z[3]);
end

endmodule
```

Simulation/Test Bench

- BCD to Excess-3 Code Converter
  - Simulation Output

x = 0000 z = 0 0 1 1
x = 0001 z = 0 1 0 0
x = 0010 z = 0 1 0 1
x = 0011 z = 0 1 1 0
x = 0100 z = 0 1 1 1
x = 0101 z = 1 0 0 0
x = 0110 z = 1 0 0 1
x = 0111 z = 1 0 1 0
x = 1000 z = 1 0 1 1
x = 1001 z = 1 1 0 0

Verilog Coding Tip

- Guildelines
  - Guideline #1: When modeling sequential logic, use nonblocking assignments.
  - Guideline #2: When modeling latches, use nonblocking assignments.
  - Guideline #3: When modeling combinational logic with an "always" block, use blocking assignments.
  - Guideline #4: When modeling both sequential and combinational logic within the same "always" block, use nonblocking assignments.
  - Guideline #5: Do not mix blocking and nonblocking assignments in the same "always" block.
  - Guideline #6: Do not make assignments to the same variable from more than one "always" block.
  - Guideline #7: Use $strobe to display values that have been assigned using nonblocking assignments.
  - Guideline #8: Do not make assignments using #0 delays.